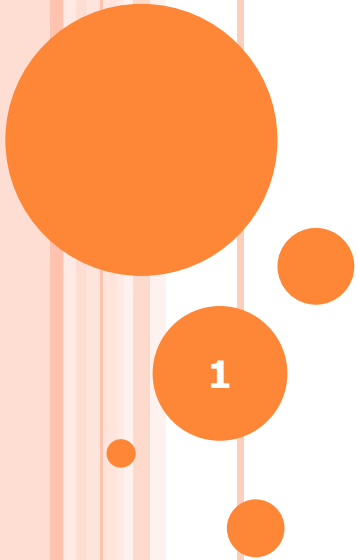
Licence STS

Université Claude Bernard Lyon I

LIFAP1 : ALGORITHMIQUE ET PROGRAMMATION IMPÉRATIVE, INITIATION



COURS 8 : Structures

OBJECTIFS DE LA SÉANCE

- Comprendre l'intérêt des structures
- Apprendre comment les manipuler

Plan

- Les structures
  - Définition
  - Intérêt
  - Syntaxe
  - Manipulation

STRUCTURE : DÉFINITION ET VOCABULAIRE

- Agrégat d'informations associées à une entité
- **Type complexe** construit à l'aide de type simples ou d'autres types complexes
- Chacune des informations contenue dans une structure s'appelle un **champ**
- Une variable de type structure est aussi appelée un **enregistrement**
  - Analogie avec les bases de données

DÉCLARATION

- En Algorithmique

```
Structure Nom_Structure
    champ1 : type
    champ2 : type
    …
Fin structure
```

- En C

```
Struct Nom_Structure
{
    type champ1;
    type champ2;
    …
};
```

EXEMPLE : EN ALGORITHMIQUE

```
Structure IdentiteEtudiant
  prenom : tableau[64] de
       caractères
  nom : tableau[64] de
       caractères
Fin structure

Structure Etudiant
  identite : IdentiteEtudiant
  note : tableau[10] de réels
  numero : entier
Fin Structure
```

- `identite`, `note` et `numero` sont les champs de la structure `Etudiant`.
- Chacun des champs est
  - soit de type simple
    - Nombre entier ou réel
  - soit de type complexe
    - `IdentiteEtudiant`

EXEMPLE : EN C

```
struct IdentiteEtudiant
{
    char prenom[64];
    char nom[64];
};

struct Etudiant
{
    struct IdentiteEtudiant identite ;
    float note[10];
    int numero;
};
```

- Mot clé : **struct**
- En C on termine la définition de la structure par un ";" après l'accolade
- Tous les champs se terminent par un ";"

UTILISATION DE CONSTANTES EN C

Possibilité de définir des constantes et de fixer leurs valeurs

```
const int longueurNom = 64 ;
const int nombreDeNotes = 10 ;

struct etudiant
{
  char nom[longueurNom] ;
  float note[nombreDeNotes] ;
} ;
```

DÉCLARATION D'UNE VARIABLE DE TYPE STRUCTURE

- Nécessaire avant d'utiliser la structure
- De même qu'on écrit "int i" avant d'utiliser "i",
  on déclare une variable de type structure Nom_Structure avant de l'utiliser

- En algorithmique :
  - Etu : etudiant
- En C :
  - struct etudiant toto ;

ACCÈS À UN CHAMP

- Pour remplir une variable de type structure,
  il faut procéder champ par champ (pas de remplissage global)
  car les types des champs sont différents

- Exemple en C:

```
struct etudiant e;          déclaration de e,
                                  variable de type
  etudiant
Cin >> e.numero;            lit le numero de
                                  l'étudiant e
cout << e.note[i];          affiche la ieme note de
                            l'étudiant e
Cin >> e.identite.nom;      avec un champ de type
                            structure
```

UTILISATION DES STRUCTURES

- Une fonction peut retourner une structure

- Une structure peut faire l'objet d'une affectation (avec une variable de même type !)

```
Etudiant e1,e2;
e2=e1;
```

- Les tableaux de structures sont possibles

```
struct etudiant classe[20] ;   /*tableau de 20
                                          etudiants*/

struct etudiant y ;
y = creerEtudiant() ;
classe[0] = y ;
```

UTILISATION

- Exemple de création d'une fiche étudiant :

```
struct etudiant creerEtudiant(void)
{
  struct etudiant e ;
  int i ;

  cout << endl << "entrer le nom :" << endl ;
  cin >> e.identite.nom ;
  cout << endl << "entrer le prénom :" << endl ;
  cin >> e.identite.prenom ;
  cout << endl << "entrer le numero de l etudiant :" << endl ;
  cin >> e.numero ;
  for (i=0; i < nombreDeNotes; i++) {
    cout << endl << "entrer la " << i << "ème note :" << endl;
    cin >> e.note[i] ;
  }
  return e ;
}
```

TRANSFORMATION

- Il est possible de transformer la fonction précédente en procédure

- L'entête devient alors :
  - `void creerEtudiant(struct etudiant & e)`

- Une structure peut être passée en donnée – résultat
- Une structure peut être retournée par une fonction

Autre exemple : RÉSOLUTION D'UN POLYNÔME

- Informations à connaître ou à évaluer
  - Les coefficients du polynôme : a, b, c donnés par l'utilisateur
  - Le discriminant delta calculé en fonction de a, b, et c
  - Le nombre de racine (en fonction de delta 0 1 ou 2 racines)
  - Les racines réelles dans la mesure où elles existent
- Soit on utilise 7 variables différentes
- Soit on met toutes ces informations dans une structure

La structure "polynôme"

- En algo

```
Structure polynome
    a,b,c : réels
    delta : réel
    nb_racines : entiers
    rac1,rac2 : réels
Fin Structure
```

- En C

```
Struct polynome
{
    float a,b,c;
    float delta;
    int nb_racines;
    double rac1,rac2;
};
```

LES FONCTIONS ASSOCIÉES

- Plusieurs fonctions à écrire
  - Saisie des coefficients
  - Calcul de delta
  - Calcul du résultat
  - Affichage du résultat

- 1 paramètre unique à passer :
  une variable de type "polynome"
- Certains champs seront remplis / calculés / affichés
- Structure passée en donnée / résultat
  ou retournée en résultat

LA FONCTION DE SAISIE

- On demande à l'utilisateur de donner les 3 coefficients a, b et c

```
struct polynome saisie_coefficients(void)
{
    struct polynome p;
    cout << "donnez a, b et c" ;
    cin>>p.a>>p.b>>p.c;
    return p;
}
```

- On crée la structure avant de la retourner car elle est n'existe pas au départ

LA FONCTION DE CALCUL DE DELTA

- On calcule delta en fonction de a, b et c

```
void calcul_delta(struct polynome & p)
{
        p.delta = (p.b*p.b) - 4*p.a*p.c;
}
```

- p est passé en donnée/resultat
  car on va utiliser 3 champs pour en remplir un.

LA FONCTION DE CALCUL DES RACINES

- On calcule les racines en fonction de delta, a, b et c

```
void calcul_racines(struct polynome & p)
{
        if (p.delta == 0)
        {
                p.rac1=-p.b / (2*p.a);
                p.rac2 =-p.b / (2*p.a);
                p.nb_racines=1;
        }
        else if (p.delta > 0)
                {
                        p.rac1=(-p.b + sqrt(p.delta))/ (2*p.a);
                        p.rac2 =(-p.b - sqrt(p.delta))/
(2*p.a);
                        p.nb_racines=2;
                }
                else p.nb_racines=0
}
```

Conclusion

- Stuctures :
  - Permettent de ranger dans une même variable toutes les informations relatives à un objet : exemple étudiant
  - Moins de variables, informations mieux organisées
  - Possibilité de faire des tableaux de structures